МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра систем штучного інтелекту

Лабораторна робота №5

з курсу “Дискретна математика ”

Виконав:
ст. гр. КН-110
Петровський Олександр

Викладач:
Мельникова Н.І.

Львів – 2018

Тема:

«Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи»

Мета роботи:

Набуття практичних вмінь та навичок з використання алгоритму Дейкстри

Теоретичні відомості:

Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G. Для розв'язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри.

«Жадібними» називаються алгоритми, які на кожному кроці вибирають оптимальний із можливих варіантів.

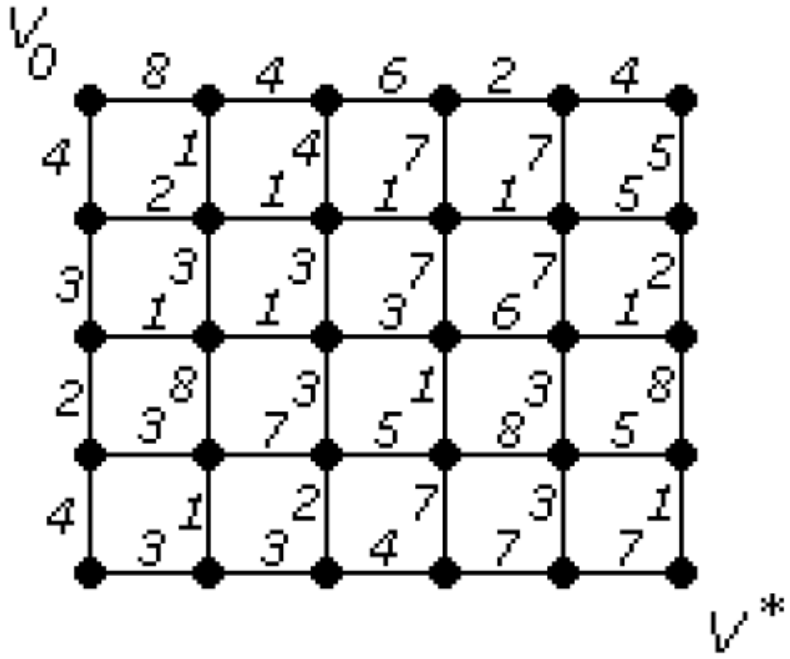
Плоским графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини.

Граф називається планарним, якщо він є ізоморфним плоскому графу. Гранню плоского графа називається максимальна по включенню множина точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребра графа. Границею грані будемо вважати множину вершин і ребер, що належать цій грані.

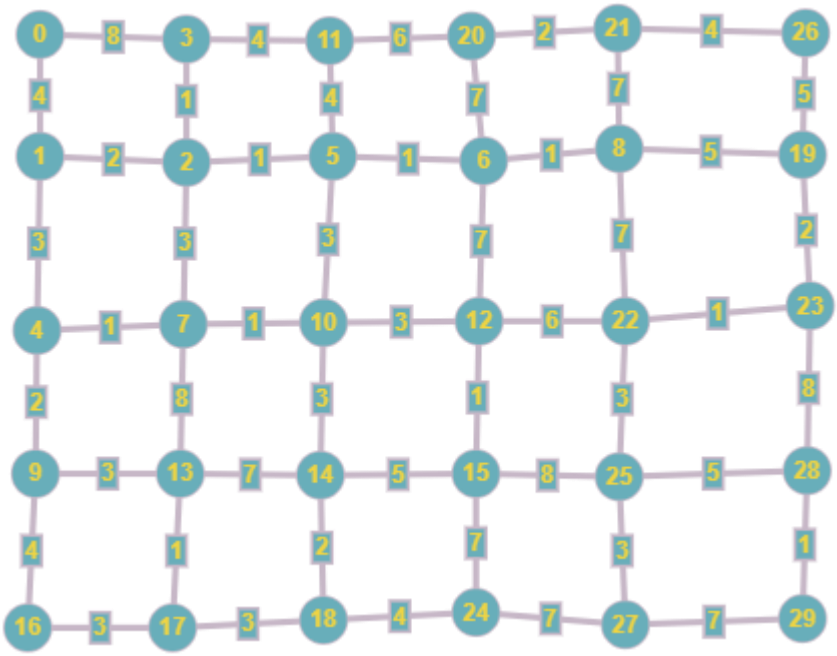
Алгоритм γ укладання графа G являє собою процес послідовного приєднання до деякого укладеного підграфа $\tilde{G}$ графа G нового ланцюга, обидва кінці якого належать $\tilde{G}$. При цьому в якості початкового плоского графа $\tilde{G}$ вибирається будь-який простий цикл графа G. Процес продовжується доти, поки не буде побудовано плоский граф, ізоморфний графові G, або приєднання деякого ланцюга виявиться неможливим. В останньому випадку граф G не є планарним.

Варіант № 5

Завдання 1:

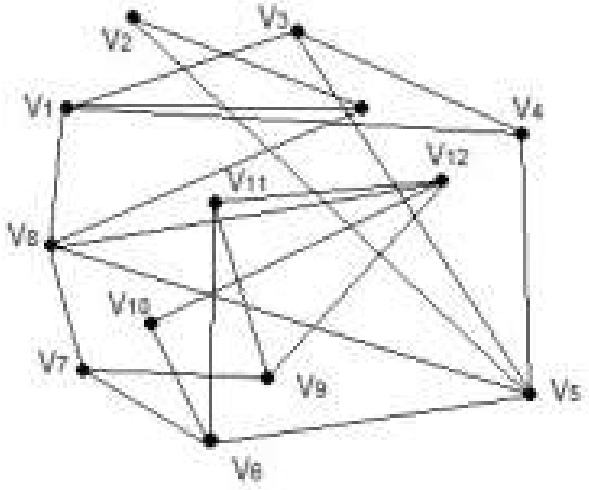


За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин V0 і V*.



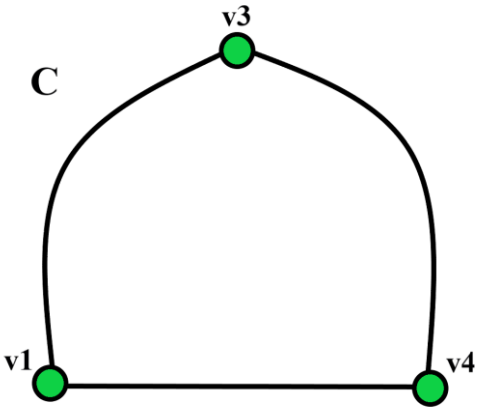
l(v1)=4, l(v2)=6, l(v3)=7, l(v4)=7, l(v5)=7, l(v6)=8, l(v7)=8, l(v8)=9, l(v9)=9, l(v10)=9, l(v11)=11, l(v12)=12, l(v13)=12, l(v14)=12, l(v15)=13, l(v16)=13, l(v17)=14, l(v18)=14, l(v19)=14, l(v20)=15, l(v21)=16, l(v22)=16, l(v23)=17, l(v24)=18, l(v25)=19, l(v26)=19, l(v27)=22, l(v28)=24, l(v29)=25
Шуканий найкоротший ланцюг: [v0, v1, v5, v6, v8, v22, v25, v28, v∗], довжина ланцюга l =l(v∗)=25.

Завдання 2:

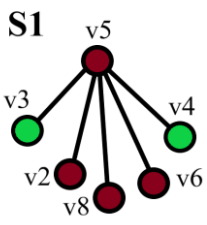


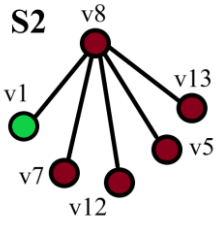
За допомогою γ -алгоритма зробити укладку графа у площині, або довести що вона неможлива.

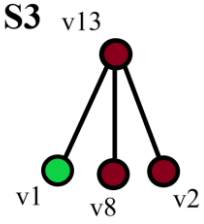
Укладемо цикл C=[1, 3, 4, 1]:

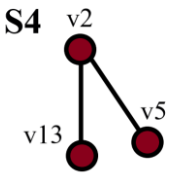


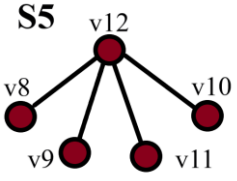
S1, S2, S3, S4, S5, S6, S7, S8, S9, S10 – сегменти графу:

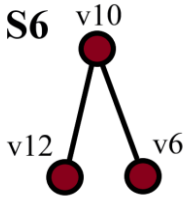


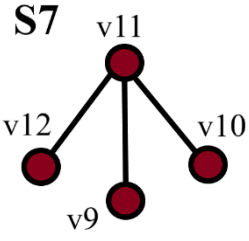


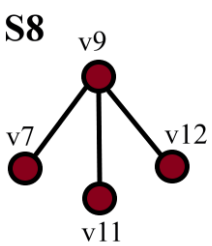


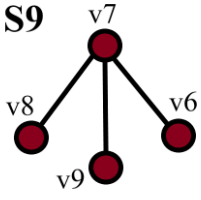


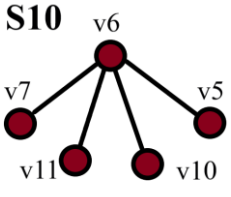




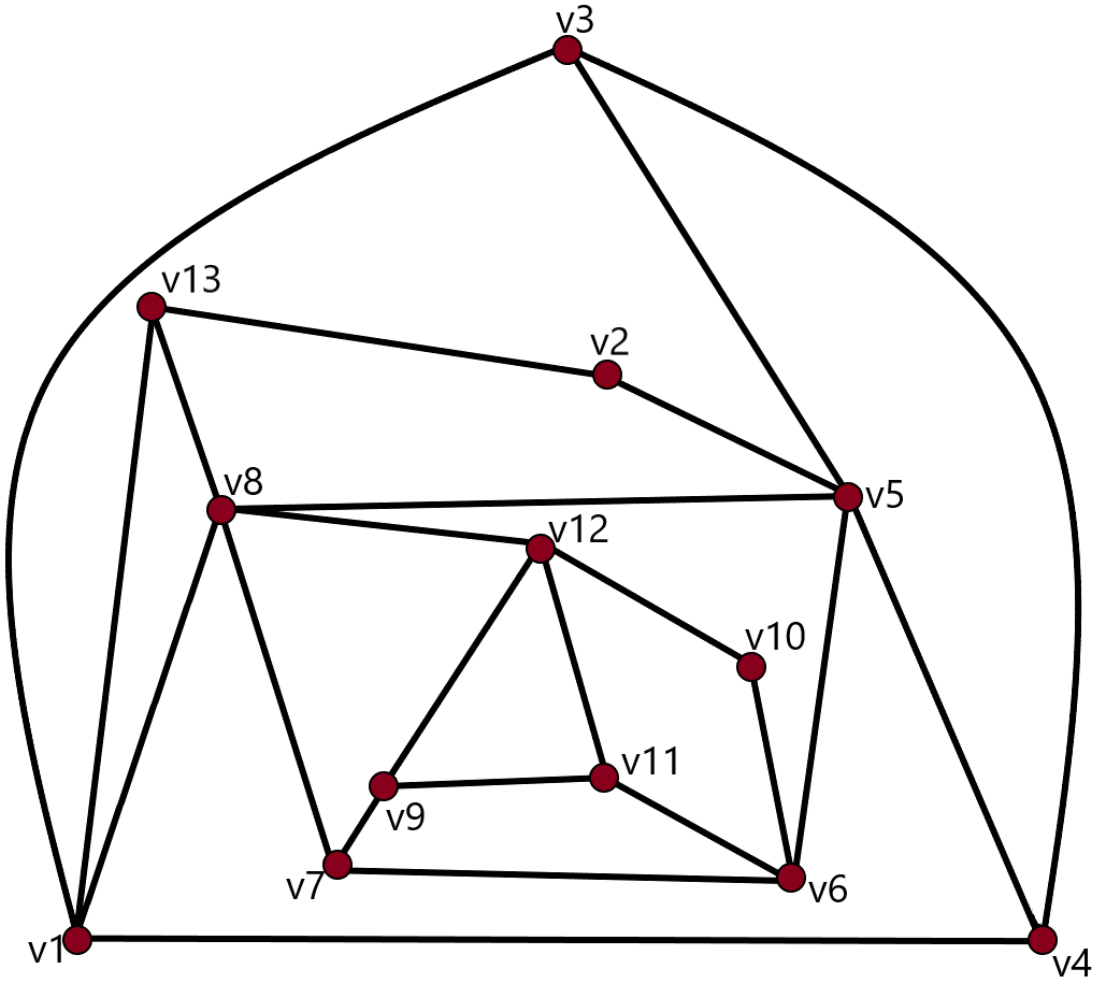






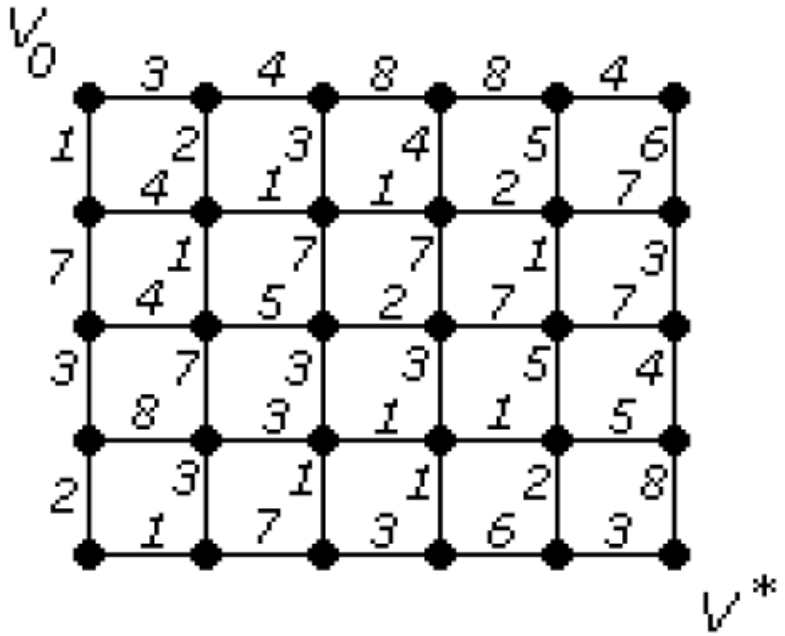


Укладемо сегменти, уникаючи перетинів:



Програма:

Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на наступному графі:



Код:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

//VARS////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////

short ver_am = 0;

//STRUCTURES//////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////

struct vert{
    short id;
    short degr;
    short used;
    short way;
    struct edge *edge[10];

};

struct edge{
    short v1id;
    short v2id;
    short wght;
    short used;
    struct vert *v1pt;
    struct vert *v2pt;
};
```

```
//PROTOTYPES////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

struct vert *create();

void connect(struct vert *a, struct vert *b, short weight);

void start_with(struct vert *vert);

void dijkstra(struct vert *vert);

//MAIN//////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int main(){

//creating the vertices
struct vert *v0 = create();
struct vert *v1 = create();
struct vert *v2 = create();
struct vert *v3 = create();
struct vert *v4 = create();
struct vert *v5 = create();
struct vert *v6 = create();
struct vert *v7 = create();
struct vert *v8 = create();
struct vert *v9 = create();
struct vert *v10 = create();
struct vert *v11 = create();
struct vert *v12 = create();
struct vert *v13 = create();
struct vert *v14 = create();
```

```
struct vert *v15 = create();
struct vert *v16 = create();
struct vert *v17 = create();
struct vert *v18 = create();
struct vert *v19 = create();
struct vert *v20 = create();
struct vert *v21 = create();
struct vert *v22 = create();
struct vert *v23 = create();
struct vert *v24 = create();
struct vert *v25 = create();
struct vert *v26 = create();
struct vert *v27 = create();
struct vert *v28 = create();
struct vert *v29 = create();

connect(v0,v1,1);
connect(v0,v2,3);
connect(v1,v3,4);
connect(v1,v8,7);
connect(v2,v6,4);
connect(v2,v3,2);
connect(v8,v5,4);
connect(v8,v12,3);
connect(v12,v15,8);
connect(v12,v14,2);
connect(v14,v17,1);
connect(v3,v4,1);
connect(v3,v5,1);
connect(v5,v13,5);
connect(v5,v15,7);
```

```
connect(v15,v18,3);
connect(v15,v17,3);
connect(v17,v20,7);
connect(v6,v11,8);
connect(v6,v4,3);
connect(v4,v7,1);
connect(v4,v13,7);
connect(v13,v16,2);
connect(v13,v18,3);
connect(v18,v21,1);
connect(v18,v20,1);
connect(v20,v23,3);
connect(v11,v19,8);
connect(v11,v7,4);
connect(v7,v9,2);
connect(v7,v16,7);
connect(v16,v10,7);
connect(v16,v21,3);
connect(v21,v22,1);
connect(v21,v23,1);
connect(v23,v25,6);
connect(v19,v27,4);
connect(v19,v9,5);
connect(v9,v24,7);
connect(v9,v10,1);
connect(v10,v26,7);
connect(v10,v22,5);
connect(v22,v28,5);
connect(v22,v25,1);
connect(v25,v29,1);
connect(v27,v24,6);
```

```
connect(v24,v26,3);
connect(v26,v28,4);
connect(v28,v29,8);

start_with(v0);

printf("\nThe shortest way is \x1b[32m%i\x1b[0m\n\n", v29->way);

}

//FUNCTIONS////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

//function for creating vertices
struct vert *create(){

    struct vert *vert = (struct vert*) malloc(sizeof(struct vert));

    vert->id = ver_am;
    vert->degr = 0;
    vert->used = 0;
    vert->way = SHRT_MAX;

    ver_am++;

    return vert;

}

//function for creating edges
void connect(struct vert *a, struct vert *b, short weight){
```

```
    struct edge *edge = (struct edge*) malloc(sizeof(struct edge));

    edge->v1id = a->id;
    edge->v2id = b->id;
    edge->wght = weight;

    edge->v1pt = a;
    edge->v2pt = b;

    a->edge[a->degr] = edge;
    a->degr++;

    b->edge[b->degr] = edge;
    b->degr++;

}

void start_with(struct vert *vert){

    vert->way = 0;
    dijkstra(vert) ;

}

void dijkstra(struct vert *vert){

    vert->used = 1;

    for (short i = 0; i < vert->degr; i++){
        if (vert->edge[i]->v2pt->used == 0){
```

```
            if (vert->way + vert->edge[i]->wght < vert->edge[i]->v2pt->way) vert->edge[i]->v2pt->way = vert->way + vert->edge[i]->wght;
        } else if (vert->edge[i]->v1pt->used == 0){
            if (vert->way + vert->edge[i]->wght < vert->edge[i]->v1pt->way) vert->edge[i]->v1pt->way = vert->way + vert->edge[i]->wght;
        }
    }

    for (short i = 0; i < vert->degr; i++){
        if (vert->edge[i]->v2pt->used == 0) dijkstra(vert->edge[i]->v2pt);
        if (vert->edge[i]->v1pt->used == 0) dijkstra(vert->edge[i]->v1pt);
    }

}
```

Висновки:

Я набув практичних вмінь та навичок з використання алгоритму Дейкстри.